Annukka Onne

Uuden ohjelmistokehyksen kehitys ja käyttöönotto

Metropolia Ammattikorkeakoulu

Insinööri (AMK)

Mediatekniikan koulutusohjelma

Insinöörityö

27.4.2016

Metropolia

Tekijä Otsikko Sivumäärä Aika	Annukka Onne Uuden ohjelmistokehyksen kehitys ja käyttöönotto 32 sivua 27.4.2016
Tutkinto	Insinööri (AMK)
Koulutusohjelma	Mediatekniikka
Suuntautumisvaihtoehto	Digitaalinen media
Ohjaaja	Lehtori Ilkka Kylmäniemi

Insinöörityön tavoitteena oli tutkia Angular-ohjelmistokehyksen uutta versiota ja sen käyttöönoton mahdollisuuksia, vaikka se on vielä kehitysvaiheessa ja beetajulkaisuversiossa. Tätä tutkimustyötä käytettiin hyväksi työn alla olleessa verkkosivustouudistuksessa ja pohdintana ohjelmistokehyksen valinnassa.

Tutkinnassa perehdyttiin Angular 2.0:n ohjelmallisiin toimintoihin ja arkkitehtuurimalleihin koodiesimerkein. Sitä verrattiin saman ohjelmistokehyksen vanhempaan versioon ja hieman muihin vastaaviin kehyksiin. Kehitteillä olevan version käyttöönoton ongelmien seurauksia analysoitiin, samoin sen tuomia mahdollisia parannuksia nykyaikaiseen ja dynaamiseen verkkokehittelyyn. Ohjelmistokehyksen toimivuutta testattiin käytännössä ja tehtiin huomioita eri vaiheissa verkkosivuston päivitystä.

Lopputuloksena todettiin ohjelmoinnin helpottuneen Angular 2.0:n käyttöönoton myötä ja huomattiin, että verkkosivuston käyttökokemus parani. Pohdintana huomioitiin, että MVC-arkkitehtuuria käyttäen myös palvelinpuolen ohjelmointia olisi syytä parantaa dynaamisempaan muotoon, jotta kokonaisuus tiedonsiirrossa olisi saumatonta. Pohdinnoissa päädyttiin toteamaan, että Angular 2.0 edustaa tulevaisuuden verkkokehitystä ja sitä käyttämällä saadaan aikaiseksi nopeita ja kevyitä kokonaisuuksia myös mobiileille alustoille tarkoitetuissa käyttöliittymissä.

Verkko-ohjelmointi kehittyy nopeasti ja vaikka päädyttiin siihen tulokseen, että Angular 2.0:n käyttöönottoa kannattaa hieman jarrutella, kunnes se on täysin stabiili, kannattaa oma verkkokehittely jo aloittaa tällä uudella versiolla, sillä vanhemmat versiot ovat pian vanhentuneita ja vailla teknistä tukea.

Avainsanat	Angular 2.0, ohjelmistokehys, JavaScript, selainpohjainen sovellus, TypeScript, dynaaminen ohjelmisto

Author	Annukka Onne
Title	The development and implementation of the new web framework
Number of Pages	32 pages
Date	27 April 2016
Degree	Bachelor of Engineering
Degree Programme	Media Technology
Specialisation option	Digital Media
Instructor	Ilkka Kylmäniemi, Senior Lecturer

The goal of this final year project was to study the new version of the Angular web framework and the deployment possibilities even though it still is in the beta phase of development. This study was used for the webpage update that was in progress and in the process of choosing the framework to be used.

In this study the code examples of the script functions and the architectural models of the Angular 2.0 web framework were investigated. The framework was compared to its older version and its competitors. Analyses of the implementation problems of the newer version and of the chances to build a modern dynamic web application were made. The functionality of the framework was tested in practice and observations were made in different stages of the webpage update.

The end results show that programming got easier with the usage of Angular 2.0 and it was realized that the user experience of the webpage improved. It was concluded that with the use of the MVC architecture the server side programming could be improved making it more dynamic so data transfer would be seamless. The end results also indicate that Angular 2.0 represents the future of web development and with it fast and light entities can be accomplished also for mobile platform user interfaces.

Web programming is developing rapidly but the study suggests that new publications with Angular 2.0 should be postponed until the framework is stable. However, development with the new version should be considered because the older versions could soon be outdated and without technical support.

Keywords	Angular 2.0, framework, JavaScript, web application, TypeScript, dynamic software

Sisällys

Lyhenteet

API	Application Programming Interface. Ohjelmointirajapinta, jonka avulla voidaan välittää informaatiota eri ohjelmien välillä.
DOM	Document Object Model. Puumalliin rakennettu hierarkkinen dokumenttioliomalli HTML-, XML- tai XHTM-kielellä kirjoitettuna.
HTML	Hypertext Markup Language. Hypertekstin merkintäkieli, joka on standardoitu kuvauskieli verkkosivujen esittämiseen.
JavaScript	Dynaamisen informaation esittämiseen käytetty komentosarjakieli verkkosivujen esittämisessä.
JSON	JavaScript Object Notation. Objektitiedostomuoto informaation välittämiseen ja tietojenkäsittelyyn.
MVC	Model-View-Controller. Malli-näkymä-käsittelijämuotoinen ohjelmistoarkkitehtuurityyli, joka erottelee järjestelmän tietojen ylläpitoon, näkymän käsittelyyn ja käskyjen ohjaamiseen.
PHP	Verkkokehittelyyn käytetty palvelinpuolen ohjelmointikieli, jota käytetään upotettuna HTML-tunnisteiden joukkoon ja jonka kirjastoissa on tuki tietokantojen käyttöön.
RESTful	Representational State Transfer. Ohjelmointirajapintojen http-protokollaa käyttävä ohjelma-arkkitehtuurimalli.
SQL	Structured Query Language. Standardisoitu relaatiotietokannan kyselykieli, jolla voi tehdään kantaan hakuja, muutoksia ja lisäyksiä.
TypeScript	Microsoftin kehittämä ohjelmointikieli, joka on laajennettu joukko JavaScript-ohjelmointia.

1 Johdanto

Tietotekniikka verkkosovellusten kehittämiseksi uudistuu nopeasti. Uusia järjestelmiä ja kehyksiä julkaistaan paljon. Mobiilit palvelut ovat tuoneet oman lisänsä verkkokehittelyyn. Tämän insinöörityön tavoitteena on tutkia uutta tekniikkaa ja sen mahdollisuuksia perinteisen verkkosivuston päivityksessä monialustaiseksi ja tehokkaaksi kokonaisuudeksi. Insinöörityössä perehdytään TypeScript-pohjaiseen beetavaiheessa olevan Angular 2.0 -ohjelmistokehityksen käyttöönoton mahdollisuuksiin ja haasteisiin. Työssä pohditaan ohjelmistokehyksen vanhemman version ja uuden testiversion käyttöönoton välistä ongelmatilannetta, koska versioiden välillä on oleellisia eroavaisuuksia ja käyttötekniikkaa.

Verkkokehittäminen on muuttunut jatkuvasti. Angular 2.0 vastaa tulevaisuuden koodikielimallia ohjelmistokehityksessä. Se on rakennettu käytettäväksi uusien standardisoitujen komentosarjakielien mukaan, vaikka ne eivät kaikkialla vielä toimisikaan. Angular 2.0 lupaa nopeampaa suoritustehoa ja latausaikaa kuin Angularin edelliset versiot. Kuitenkaan ohjelmistokehyksellä ei ole vielä loppuun asti vietyä dokumentaatiota ja ohjelmointivirheitä, eli bugeja, löytyy edelleen uudestakin versiosta. Houkutuksena on kuitenkin käyttää tätä uusinta teknologiaa, sillä edellinen versio voi vanhentua nopeasti, ja näin ollen työn alla olevasta sivuston päivitystyössä säästyttäisiin turhalta versiopäivitykseltä. Tiedot Angularin uudesta versiosta, kritiikistä huolimatta, vaikuttavat lupaavilta.

Insinöörityössä perehdytään Angular 2.0:n uusiin ominaisuuksiin ja mallinnetaan sen oleellisimpia käyttötarkoituksia. Aluksi käydään pääpiirteittäin läpi Angularin historiaa ja sen käyttöä koko sen olemassaolon aikana ja verrataan hieman muita vastaavia ohjelmistokehyksiä. Luvussa 3 perehdytään Angular 2.0:n arkkitehtuurisiin pääpiirteisiin ja avataan ohjelmoinnin malleja käymällä läpi koodiesimerkkejä Angular 2.0:n toiminnoista yleisellä tasolla. Luvussa 4 käydään läpi, kuinka Angular 2.0 -ohjelmistokehystä on käytetty päivityksen alla olevalla verkkosivustolla, jota varten myös tätä tutkimustyötä tehdään. Samassa luvussa analysoidaan myös ohjelmistokehyksen käytön hyötyjä ja haittoja ja pohditaan sen tulevaisuutta. Lopputuloksena ratkaistaan, onko Angular 2.0 vielä valmis käyttöönotettavaksi ja minkälaisia uusia ulottuvuuksia se tuo tulevaisuuden verkkokehitykseen.

2 Angular-ohjelmistokehys

2.1 Angularin historia

Angular-ohjelmistokehys on kehitetty, jotta saataisiin aikaiseksi helppo ja nopeakäyttöinen järjestelmä dynaamisten verkkosivustojen aikaansaamiseksi. Kuten Angularin kehittelijä Miško Hevery [1] on sanonut, Angularin tarkoitus on tehdä verkkoselaimesta älykkäämpi. Hän on monesti myös todennut, että hänen on vaikea selittää, mikä Angular oikeastaan on. Se ei ole varsinaisesti ohjelmistokehys eikä perinteinen tietotekninen kirjasto. Hevery myös toteaa, että hän halusi vain helpottaa modernia verkkokehitystä ja tehdä siitä yksinkertaisempaa niillekin, jotka eivät ymmärrä monimutkaista palvelinohjelmointia [1; 2].

Angular on nykyään Googlen ylläpitämä ohjelmistokehys. Se on alun perin kahden henkilön, Miško Heveryn ja Adam Abronsin, kehittämä. Nykyisin sillä on GitHub-versionhallintajärjestelmäsivustolla yli 200 kehittäjäjäsentä [3], ja ohjelmistokehyksen oman sivuston mukaan Angular 2.0:n parissa pääasiallisesti työskentelee seitsemän henkilöä, myös alkuperäissuunnittelija Miško Hevery ja Googlen puolelta 21 muuta henkilöä ottaa osaa kehittelyyn [4]. Angularia kehittämässä on myös hyvin aktiivinen ja suuri yhteisö vapaaehtoisia ohjelmoijia. Angular on avoimen lähdekoodin ohjelmistokehys MIT-lisenssillä, eli se on vapaasti muokattavissa omaan käyttöön, ja monet tahot tekevät muutoksissa yhteistyötä myös Angular-tiimin kanssa [2].

2.2 Angular 2.0

Angularin ohjelmistokehys tarjoaa keinoja näyttää erilaista dataa verkkoselaimessa ilman, että manipuloidaan itse dokumenttioliomallia (DOM) ja ilman jatkuvia palvelupyyntöjä palvelimelta. Angular käyttää esilatausohjelmallisuutta, ja loppukäyttäjän tekemiä muutoksia objektiarvoihin tarkkaillaan jatkuvasti ohjelmallisesti. Samalla komentosarjoja, eli skriptejä, pyritään minimoimaan ja estämään niitä latautumasta näkymän eli HTML-koodin puolelta. Tämän kaiken voi rakentaa perinteisinkin menetelmin, mutta Angular-ohjelmistokehys tuo selkeyttä tähän erotteluun ja ohjelmointia voidaan rakentaa valmiiden mallien pohjilta. Ohjelmistokehyksessä on monimutkaisiakin ohjelmointimalleja, ja niihin ei tarvitse löytää kuin oikeat syötteet ja näin ohjelmistokehys huolehtii minimoidusta komentosarjamallista. Angularia voisi kutsua myös HTML-kääntäjäksi,

sillä se käyttää omia HTML-elementtien kaltaisia merkkijonoja laukaisemaan ohjelmalliset muutokset verkkoselaimessa. Luvussa 3 käydään tarkemmin läpi näitä ohjelmointimalleja ja perehdytään niiden toimintaan syvemmin.

2.3 Angular 1.x -versiot

Angular 2.0:n kehityksen syy on ollut lähinnä parantaa Angularin toimintoja ja muuttaa verkkolatausaikoja aikaisempia vanhentuneita ohjelmamalleja käyttäviä 1.x -versioita nopeammiksi. Siihen on otettu käyttöön uusimpia standardeja, ja sen käyttökoodit eroavat edellisistä versioista sen verran, että siitä päätettiin julkaista 2.0 -versio sen sijaan, että olisi jatkettu senhetkisestä 1.3 -versiosta. Myös Angular 1.x -versioita on julkaistu ja päivitetty 2.0 -testiversiojulkaisun jälkeenkin. Tällä hetkellä uusin päivitysversio on 1.5. Versioita kehitellään siis rinnakkain, kuten kuvassa 1 näkyvissä olevista versiolatauspäivistä voidaan huomata. Angularin 1.x -versioita myös pyritään päivittämään enemmän 2.0 -version suuntaan. Versioita voidaan käyttää rinnakkain, mutta 1.x -versioiden komennot eivät ole käypiä 2.0 -version kanssa. JavaScript-ohjelmointiosat toimivat molemmissa versioissa, sillä TypeScript on Microsoftin kehittelemä JavaScriptin laajennus.

1.4.7/	01-Oct-2015 00:12
1.4.8/	20-Nov-2015 08:13
1.4.9/	21-Jan-2016 13:21
1.5.0/	05-Feb-2016 12:01
1.5.0-beta.0/	01-Oct-2015 00:12
1.5.0-beta.1/	01-Oct-2015 00:12
1.5.0-beta.2/	18-Nov-2015 00:22
1.5.0-rc.0/	09-Dec-2015 14:35
1.5.0-rc.1/	15-Jan-2016 22:28
1.5.0-rc.2/	28-Jan-2016 12:26
. . .	
2.0.0-alpha.44/	16-Oct-2015 04:19
2.0.0-alpha.45/	29-Oct-2015 02:03
2.0.0-alpha.46/	11-Nov-2015 22:04
2.0.0-alpha.47/	01-Dec-2015 20:41
2.0.0-alpha.48/	05-Dec-2015 03:31
2.0.0-alpha.49/	09-Dec-2015 02:20
2.0.0-alpha.50/	09-Dec-2015 03:46
2.0.0-alpha.51/	10-Dec-2015 04:01
2.0.0-alpha.52/	10-Dec-2015 09:09
2.0.0-alpha.53/	13-Dec-2015 23:45
2.0.0-alpha.54/	15-Dec-2015 14:10
2.0.0-alpha.55/	15-Dec-2015 18:14
2.0.0-beta.0/	15-Dec-2015 19:27
2.0.0-beta.1/	08-Jan-2016 21:53
2.0.0-beta.2/	29-Jan-2016 21:53
2.0.0-beta.3/	03-Feb-2016 18:33
2.0.0-beta.4/	10-Feb-2016 23:57
2.0.0-beta.5/	11-Feb-2016 00:28
2.0.0-beta.6/	12-Feb-2016 00:17
2.0.0-beta.7/	19-Feb-2016 01:31
2.0.0-beta.8/	02-Mar-2016 21:26
2.0.0-beta.9/	10-Mar-2016 00:29

Kuva 1. Angularin kirjastot ja niiden latauspäivät [7].

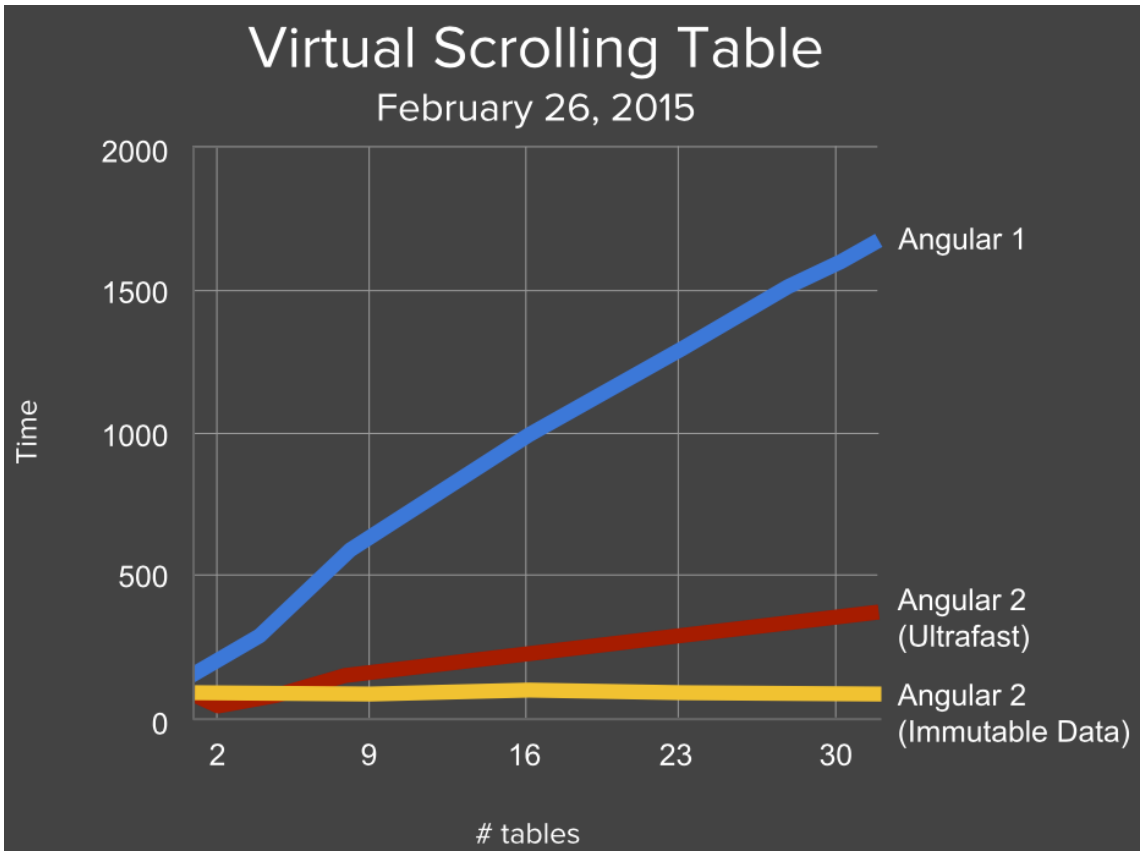
Käyttäjiä kehotetaan vielä pysymään vanhemmassa versiossa suurten verkko-ohjelmien ja ohjelmistokehyspäivitysten kohdalla. Uusien ohjelmien työstössä taas kehotetaan kokeilemaan 2.0 -versiota sen sallimilla alueilla ja samalla seuraamaan päivitysten vaillinaisia ongelmakysymyksiä ja niiden korjauksia [6]. Angularin vanhoja versioita ylläpidetään vielä noin vuoden verran, minkä jälkeen päivityksiä niihin ei enää tule. Tätä ongelmatilannetta analysoidaan enemmän luvussa 5.

Suurimmat erot Angularin 2.0 ja 1.x -versioiden välillä ovat niiden luokkarakenteen elementit. Uudessa versiossa on luovuttu *$scope*-nimikkeen käytöstä ja *controller*-funktioista. Uudessa versiossa metadata on sisällytetty jokaiseen luokkaan toimintojen kanssa uusien suunnittelumallien avulla. Toiminnot, jotka ovat myös sidottuina luokkiin, on jaettu direktiivipohjaisesti ja toiminta tapahtuu luokan komponenttien määrittelemänä. Uudessa versiossa lisätään vain asiaan liittyvät elementit niitä tarvitseviin luokkiin. Se selkeyttää ihmisluettavuutta ja näin ollen helpottaa itse ohjelmointia.

Angular 2.0 -versiossa on otettu käyttöön *import*-moduulit, eli moduulit tuodaan tapauskohtaisesti käyttöön. Tämä onnistuu käyttämällä Angular 2.0:aa TypeScript-ohjelmointikielellä. Itse Angularin lähdekoodi on siis kirjoitettu tällä TypeScript-kielellä, ja 2.0 -version ohjeistus ja dokumentaatio on viety pisimmälle tätä kieltä käyttäen. Angularia voi myös käyttää Dart- ja muilla JavaScript-ohjelmointikielillä, mutta näiden ohjeistus on vielä vaillinaista uudessa versiossa. Näin ollen myös insinöörityössä keskitytään täysin tuohon TypeScript-ohjelmointimalliin.

Uudessa versiossa on myös huomattavasti parannettu toimintanopeutta, mikä parantaa mobiilikäyttöä ja suurien tietojärjestelmien latautumista. Nopeus ei ole tärkein elementti omassa verkkosivustopäivityksessäni, mutta nopeuden parantaminen on huomattava etuus Angularin versioita vertailtaessa. Tämä on ollut myös yksi tiimin pääsyistä julkaista uusi versiomuutos. Sivulla 5 kuvassa 2 nähdään suorituskyvyn nopeuttamisen muutokset monikerroksisessa latauksessa.

Alkuperäinen ajatus tehdä dynaamisten verkko-ohjelmien rakentamisesta helpompaa on säilynyt Angularin uudessa versiossa, ja tässä osa-alueessa on pyritty myös tekemään parannuksia.



Kuva 2. Nopeusvertailu taulukkojen latauksesta yhtäaikaisesti [5].

2.4 Kilpailevat ohjelmistokehykset

Angular on noussut johtavaksi verkkosovellusten ohjelmistokehykseksi. Sitä käytetään huomattavasti enemmän kuin sen kilpailijoita. On kuitenkin hyvä huomioida muitakin vastaavia kehyksiä, jotta voidaan tarkastella Angularin puutteita ja ymmärretään sen ylivoimaisuutta.

Esimerkiksi Suomen työvoimatoimiston verkkosivun (mol.fi) avoimien työpaikkojen angular-sanahaulla työpaikkoja löytyy 53 [8], kun taas kilpailevia sovelluskehyshakusanoja käyttäen työpaikkoja löytyi huomattavasti vähemmän. Yhteensä niitä oli yli puolet vähemmän kuin sanahaulla angular. Nämä haut antavat jonkinlaista kuvaa Angular-ohjelmistokehyksen osaamisen haluttavuudesta tällä hetkellä.

Vaikka Angularia käytetäänkin ohjelmistokehyksistä eniten, se ei tarkoita, että se olisi aina paras vaihtoehto kyseessä olevalle sovellukselle. Kilpailijoita enemmän käytettynä sen huomattavin hyöty voi olla suuri yhteisöllisyys ja apu ongelmatilanteissa, vaikkei se

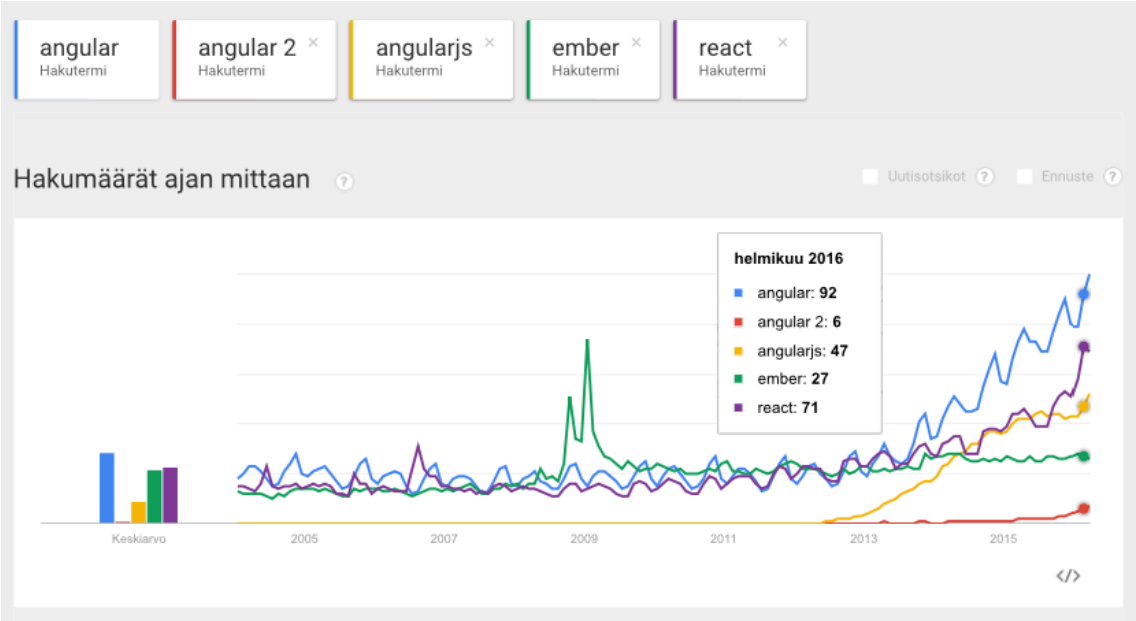
aina olisikaan yksinkertaisinta koodia tai nopeinta käyttöliittymää. Angular 2.0:n kohdalla tuki ja apu on kuitenkin vielä vähäistä, sillä suurin osa ohjelmoijista käyttää edelleen Angularin 1.x -versioita.

Taulukossa 1 ovat listattuna tärkeimmät ehdokkaat Angularin kilpailijoiksi. Siinä on eritelty niiden hyvät ja huonot ominaisuudet sekä ominaisuudet, jotka voivat olla sekä hyödyksi että haitaksi riippuen sovelluksesta.

Taulukko 1. Vertailuehdokkaat JavaScript-pohjaiselle ohjelmistokehykselle.

	Hyvät puolet	**Huonot puolet**	**Molempia**
Ember.js	• paljon valittavia lisäosia • komentorivityöskentely mahdollista • Ruby on Rails -yhteensopiva	• mobiilikäytössä sellaisenaan raskas • kevyt dokumentaatio	• valmiit ratkaisumallit • tiivis yhteisö
React.js	• virtuaali DOM • käyttäjäkunta kasvussa	• alkuvaiheessa, uusin versio 0.14 • ei palvelinpuolen kommunikointia	• panostettu näytön renderöintiin
Knockout.js	• kevyt • toimivuus vanhoilla selaimilla • .NET-yhteensopiva	• vanhentunut • ei reititystä	• sopii pieniin sivustoihin • suppea
Backbone.js	• kevyt • hyvä dokumentaatio	• tukeutuu usein kolmansien osapuolien kirjastoihin • suora DOM-manipulointi	• suppea • ei valmista rakennetta

Sivulla 7 kuvassa 3 on nähtävillä Googlen rekisteröimiä hakukonetuloksia. Angular 2 -hakuja on vielä vähän, mutta osa hauista toteutuu myös pelkällä angular-haulla, jolla on siis kärkitila tässä tilastossa. React on myös nousussa, ja monilla keskustelufoorumeilla ja blogikirjoituksissa on kehuttu Facebookin ylläpitämää React-ohjelmistokehystä Angularin ohitse [9].



Kuva 3. Googlen mittaamien verkkohakujen kehityskaavio [10].

Oikeastaan muiden sovelluskehyksien käyttöä ei alkuun tässä työssä edes harkittu. Ember olisi voinut olla paras ja kevyempi vaihtoehto projektin alla olevaan verkkosivu-päivitykseen. Toinen lähestymistapa oli kuitenkin oppia käyttämään tätä uutta Angular 2.0 -ohjelmistokehystä, sillä sen hallitseminen tuntui tärkeämmältä omalla urakehityksselläni, ja insinöörityö loi mahdollisuuden sen lähempään tutkimiseen. Näiden kehyksien peruskäytöllä ei suurta eroa ole, ja Angularin laajentuminen eri osa-alueille kiehtoi. Näin ollen verkkosivu-uudistus päätettiin tehdä alkuperäisen vaihtoehdon parissa.

3 Ohjelmointimalleja suunnitteluun

3.1 Angular 2.0:n arkkitehtuuri

Angular 2.0 -version arkkitehtuurissa on kahdeksan pääasiallista rakennetta, jotka voidaan nähdä alla olevassa luettelossa selitytettyinä. Suluissa olevat englanninkieliset termit selventävät ohjelmallisia yhteyksiä. Näitä termejä käytetään myös tämän insinöörityön ohjelmakoodiesimerkkien selityksissä.

- **Moduuli (Module):** Angular on kokonaisuudessa rakennettu erilaisista moduuleista, ja sitä kehotetaan myös kirjoittamaan moduulimaisin kokonai-

suuksin, joihin tuodaan muita moduuleita import-komennoilla, ja export-komennolla viedään vain yksi luokka muihin moduuleihin.

- **Komponentti (Component):** Komponentit täyttävät informaatiota näytölle ja välimuistiin käyttäjän toimintojen mukaan. Angular huolehtii komponenttien elinkaarista.

- **Mallinne (Template):** Mallinteet ovat HTML-pohjaista koodikieltä, joka antaa Angularille määritteet tulkita komponentit. Komponentteja voidaan kirjoittaa räätälöidysti.

- **Metatieto (Metadata):** Metatieto kertoo, miten prosessoida ohjelmallisia luokkia. @-funktiot ottavat vastaan objekteja, jotka Angular kääntää tarvittavaksi metatiedoksi ja käyttää tiedon esittämiseen oikealla tavalla.

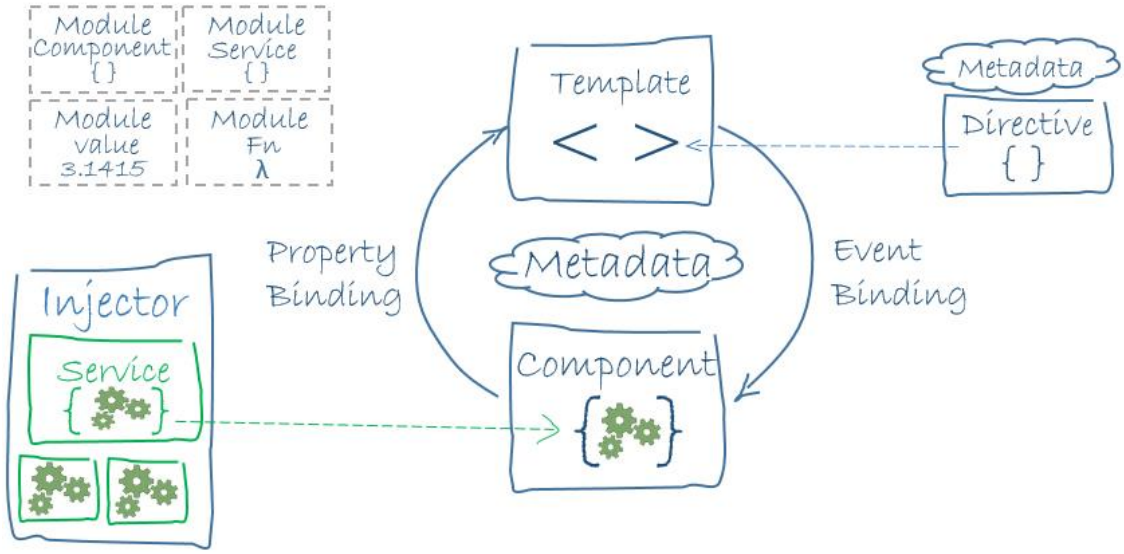
- **Tietokytkentä (Data Binding):** Angularissa on valmiina sisäänrakennettu tiedon liikkumismekanismi ohjelmallisen hierarkian mukaisesti molempiin suuntiin käyttäjän ja selaimen välillä. Sitä voidaan käyttää tiedon, ominaisuuden ja tapahtumien lisäykseen.

- **Direktiivi (Directive):** Direktiivi on metadataa välittävä luokka, joka antaa toimintaohjeita informaation täyttämiseen näyttöelementeissä. Angular sisältää valmiita direktiivejä, ja niitä voi myös rakentaa itse.

- **Palvelu (Service):** Palvelut ovat tyypillisiä luokkarakenteita, jotka ovat yleisiä missä tahansa oliotyyppisessä ohjelmointikielessä, ja Angular ei ole tästä poikkeus. Palvelut ovat siis myös Angular-tyyppisen ohjelmoinnin perusta, ja ne välittävät tiedon rakenteita muualle ohjelmaan.

- **Riippuvuusinjektio (Dependency Injection):** Riippuvuusinjektiota käytetään täyttämään komponentteja tarvittaviin palveluihin. Komponenttien muodostaja kutsutaan palveluiden mukana argumentteina.

Sivulla 9 kuvassa 4 nähdään arkkitehtuuriset kohdat piirrettynä diagrammimaiseen muotoon. Se selkeyttää eri osioiden toiminnallisuuksien yhteyksien ymmärtämistä ja käyttötapaa. Angularin arkkitehtuuriin kuuluu muitakin valinnaisia elementtejä, kuten animaatiot, komponenttireitittimet, lomakkeet ja esilatausohjelmallisuus ja niin edelleen. Seuraavissa luvuissa käydään läpi tämän insinöörityön pohjalla olevan verkkosivuston päivityksen kannalta oleellisimpia toimintoja.



Kuva 4. Angular 2.0:n arkkitehtuuridiagrammi [11].

3.2 Datan näyttäminen

Yksi Angularin perusominaisuuksista on sitoa ohjelmalliset komponentit näytettäväksi HTML-elementein. Näytettävät elementit ovat sidottuna komponentin ominaisuuksiin. Kun ominaisuuksia muutetaan, Angular päivittää selaimessa näytettävän sisällön. Tätä hyödyntämällä voidaan käyttää erilaisia objekteja informaation esittämiseen. Asiaa on hyvä käydä hieman läpi lähdekoodiesimerkein. Koodiesimerkissä 1 on yksinkertainen peruskomponentti, joka sisältää erilaisia toimivuuksia, jotka ovat ominaisia Angular 2.0 -ohjelmoinnille.

```
import {Component} from 'angular2/core';
import {Hero} from './hero';

@Component({
   selector: 'my-app',
   template: `
      <h1>{{title}}</h1>
      <h2>My favorite hero is: {{myHero.name}}</h2>
      <p>Heroes:</p>
      <ul>
         <li *ngFor="#hero of heroes">
             {{ hero.name }}
         </li>
      </ul>
      <p *ngIf="heroes.length > 3">There are many
            heroes!</p>
`
})
```

```
export class AppComponent {
   title = 'Tour of Heroes';
   heroes = [
     new Hero(1, 'Windstorm'),
     new Hero(13, 'Bombasto'),
     new Hero(15, 'Magneta'),
     new Hero(20, 'Tornado')
  ];
  myHero = this.heroes[0];
}
```

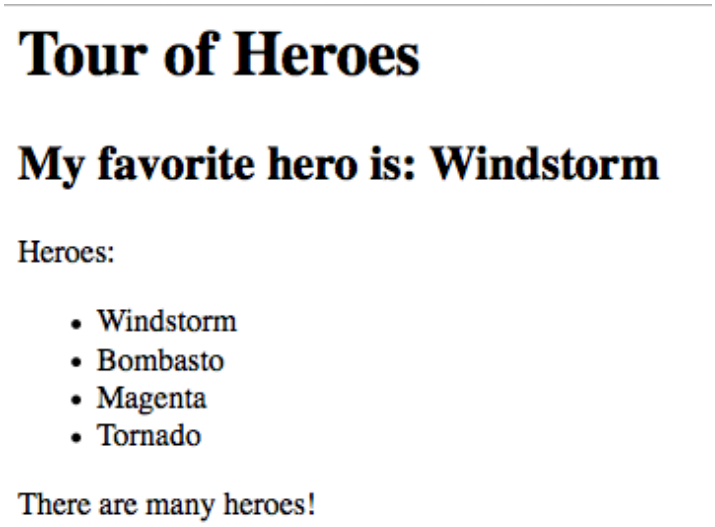
Koodiesimerkki 1. Ote Angular-komponentista [12].

Esimerkin osassa *template* on määritelty kaksinkertaisilla aaltosulkeilla muuttuvien elementtien paikkaa HTML:ssä ja luokassa *AppComponent* määritellään se informaatio, joka näihin paikkoihin on tarkoitus sisällyttää. Taulukossa *heroes* on käytetty koodiesimerkissä 2 luotua objektia luomaan sisältöä komennoilla *new Hero()*. *Import*-komennoilla on tuotu Angularin ydinkoodista *component*-luokan ohjelmallisuus ja tämä *Hero*-luokka taulukon esittämiseen.

```
export class Hero {
   constructor(
   public id:number,
   public name:string) { }
}
```

Koodiesimerkki 2. *Hero*-objektin määritelmä [12].

@Component-osion sisällä oleva *selector* ilmaisee index.html -tiedostossa määriteltyä paikkaa, jossa tämä ohjelmallisuus suoritetaan. Se on merkitty tunnisteella <my-app>. Komponentin *Template*-osiossa on käytetty myös HTML-attribuutin kaltaisesti ehtomuotoja **ngFor* ja **ngIf*, jotka ovat Angularin sisäänrakennettuja ehtolauseita. Taulukon *heroes* sisältö voidaan siis silmukoida -tunnisteiden sisälle. Viimeinen <p>-tunniste näkyy vain, jos ehtolause täyttyy (tässä tapauksessa se täyttyy). Sivulla 11 kuvassa 5 nähdään kuvakaappaus äskeisen koodin tuottama näkymä selaimessa.



Kuva 5. Koodiesimerkin 1 tuottama selainnäkymä.

3.3 Lomakkeet ja käyttäjän syötteet

Klikkien säästäminen on yksi pääsyistä, miksi tekeillä olevaan verkkosivustoon kaivattiin päivitysten yhteydessä ohjelmistokehystä. Lomakkeiden tarkastaminen ja esikatselun näyttäminen ennen tietojen päivitystä tietokantaan ovat aikaisemmin olleet erillisiä toimintoja ja uudelleenpäivitettyjä sivustoja palvelimelle ladatusta informaatiosta. Tarkoitus oli, että esikatselua voidaan nähdä samanaikaisesti tietoja syöttäessä ja kaikki sallitut merkit huomioidaan jo lomakkeita kirjoittaessa. Angularin lomakkeet ja käyttäjäsyötteet on rakennettu niin, että niitä voidaan kontrolloida, validoida, seurata ja virhejäljittää samanaikaisesti pysäyttämättä muuta ohjelmallisuutta. Koodiesimerkissä 3 käydään läpi yksinkertaisen lomakkeen toimintoja.

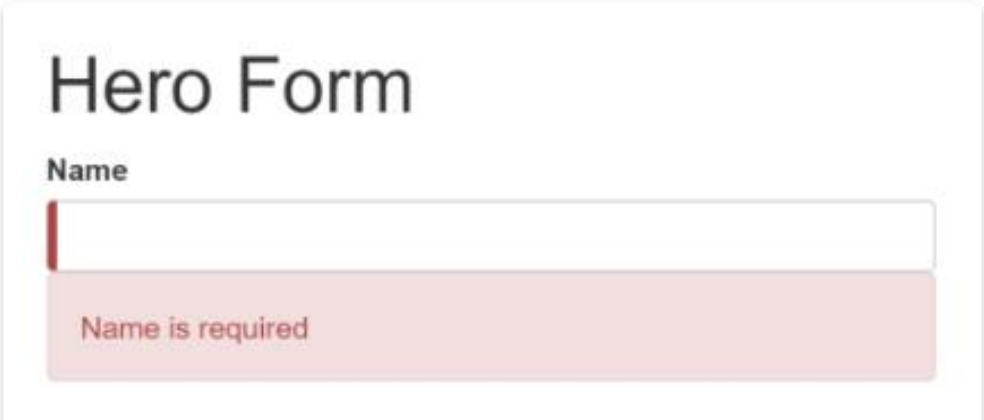
```
<div class="container">
  <div [hidden]="submitted">
    <h1>Hero Form</h1>
    <form *ngIf="active" (ngSubmit)="onSubmit()"
                #heroForm="ngForm">
    <div class="form-group">
      <label for="name">Name</label>
      <input type="text" class="form-control" required
                [(ngModel)]="model.name"
                ngControl="name" #name="ngForm" >
      <div [hidden]="name.valid || name.pristine"
                class="alert alert-danger">
           Name is required</div>
    </div>
  </div>
```

```
    <button type="submit" class="btn btn-default"
            [disabled]="!heroForm.form.valid">Submit</button>
    </form>
  </div>

  <div [hidden]="!submitted">
    <h2>You submitted the following:</h2>
    <div class="row">
    <div class="col-xs-3">Name</div>
    <div class="col-xs-9 pull-left">{{ model.name }}</div>
  </div>
 </div>
```

Koodiesimerkki 3. Lomakkeen html-mallinne [13].

Koodiesimerkissä 3 nähdään, kuinka lomake on rakennettu. HTML:n sisällä olevat (*class*) attribuutit ovat kolmannen osapuolen Twitter Bootstrap -kehyksestä, joka luo tässä visuaalista lisää näyttämään esimerkiksi syötteiden virhetilanteita. Angular ei siis estä käyttämästä ulkopuolisia tietoteknisiä kirjastoja yhtä aikaa sen kanssa. Sulkeissa ja aalto- sekä hakasulkeissa olevat elementit ovat Angulariin vaikuttavia määritteitä, kuten myös ng-alkuiset attribuuttiarvot. Tässä koodissa kerrotaan, että lomake (<form>) on sidottu Angularin lomakekontrollereihin ja siinä on tekstikenttä, joka ottaa vastaan tekstisyötteitä. Varoitusteksti tulee näkyviin aina, jos kentän arvoa muutetaan ja se ei ole validi. Paikkansapitävä ehto on koodissa sidottu kaksoispiippumerkein (||). Lomakkeen taustalla toimii Angularille ominainen komponenttiluokka, joka hoitaa ohjelmallisen datansiirron näkymästä ohjelmalle ja takaisin tähän näkymään. Vain mikäli esimerkin tekstisyöte on validi, näytetään nappi (*submit*), joka aktivoi tiedon tallentamisen komponentissa sijaitsevassa erillisessä funktiossa. Tämän jälkeen komponentti mahdollistaa alaosan tunnisteiden piilotettujen osien ([hidden]) näyttämisen uusine tietoineen. Kuvassa 6 on koodin lomakkeen tuottamasta näkymästä selaimessa.



Kuva 6. Koodiesimerkin 3 tuottamaa selainnäkymää.

3.4 Reititin

Reititin on olennainen osa, kun rakennetaan yksisivuista ohjelmaa useilla eri näkymillä. Reititin hoitaa navigointia ja komponenttien välisiä linkkejä. Omassa ohjelmassani näkymiä on useita ja alkuperäisesti ne toimivat yhden lähdekooditiedoston sisällä. Tarkoitus on pilkkoa ne helpommin käsiteltäviksi kokonaisuuksiksi, ja Angularin käyttö mahdollisti tämän helpolla tavalla. Koodiesimerkin 4 myötä käydään läpi reitittimen eri osa-alueita.

```
...
@Component({
   selector: 'my-app',
    template: `
      <h1 class="title">Component Router</h1>
      <nav>
       <a [routerLink]="['CrisisCenter']">Crisis Center</a>
       <a [routerLink]="['Heroes']">Heroes</a>
      </nav>
      <router-outlet></router-outlet>
   `,
   providers: [DialogService, HeroService],
   directives: [ROUTER_DIRECTIVES]
})
@RouteConfig([
   { // Crisis Center child route
     path: '/crisis-center/...',
     name: 'CrisisCenter',
           component: CrisisCenterComponent,
     useAsDefault: true
   },
   {path: '/heroes', name: 'Heroes',
     component: HeroListComponent},
   {path: '/hero/:id', name: 'HeroDetail',
     component: HeroDetailComponent},
   {path: '/disaster', name: 'Asteroid',
      redirectTo: ['CrisisCenter', 'CrisisDetail', {id:3}]}
])
export class AppComponent { }
```

Koodiesimerkki 4. Osa reititinkomponentin lähdekoodista [14].

Komponentin *template*-osiossa on määritelty <a>-tunnistein klikattavia linkkejä, joista Angular tunnistaa [router-link]-ominaisuudella merkityn määritteen ja hakee reitittimen vaihtoehdon *RouteConfig*-asetuksista. Nämä asetukset kertovat halutun selainpolun, nimikkeen ja sen komponentin, joka tuottaa kyseisen näkymän. Tunniste <router-outlet> taas määrittää, mihin alinäkymät sijoitetaan. Komponentille pitää myös kertoa, että siinä käytetään tätä Angularin reititinkirjastoa, joka siis on erillinen kirjasto Angularin ydinkirjastosta (*directives: [ROUTER_DIRECTIVES]*). Kirjastot on ladattu tässä *in-*

dex.html -tiedostoon ja tuotu *import*-komennoilla moduuliin. Koodiesimerkissä import-komennot on jätetty pois kolmen pisteen kohdalta. Myös jokainen reitittimen käyttämä näkymäkomponentti on sisälletty *import*-komennoilla tähän moduuliin. Reititinkirjastot on myös huomioitu ohjelman esilatausvaiheessa *Bootstrap*-metodissa. Reititinkirjasto on Angularin erillinen palveluosa (*service*).

3.5 Mallinne

Angularissa mallinne (engl. template) on pääosassa näkymän eli ulkoasun muodostamisessa MVC-tyyppisessä (Model-View-Contoller, suom. malli-näkymä-käsittelijä) arkkitehtuurissa. Angular tosin ei toimi pelkästään MVC-arkkitehtuurin mukaisesti, vaan sitä voi käyttää myös rakennettaessa MVVM-tyyppisiä (Model-View-ViewModel) sovelluksia. Igon Minar, yksi Angularin pitkäaikaisista kehittelijöistä, on jo alkuaikoina julistanut, että Angular toimii pikemminkin MVW-tyylillä (Model-View-Whatever), jossa whatever (suom. mikä tahansa) voi olla siis mikä tahansa, minkä ohjelmoija parhaaksi näkee [15].

Näkymä on kuitenkin osa-alue, joka Angularissa pyritään rakentamaan pääpiirteissään HTML-kielellä. Angularilla on omia komponentteja, jotka täydentävät perinteisiä HTML-merkintöjä. Vaikka Angular 2.0 käyttää näitä omia HTML:n ulkopuolisia tunnisteita, attribuuttien kaltaisia elementtejä ja luokkia, se tuottaa aivan validia lähdekoodia selaimeen [16]. Taulukossa 2 nähdään Angularissa käytettävät tietokytkentämallit. Näissä sidoksissa käytetään vain ominaisuuksia ja tapahtumia (*events*).

Taulukko 2. Tietosidontaesimerkit mallinteiden päivittämiseksi [17].

Sidonnan tyyppi	**Kohde**	**Esimerkki**

Ominaisuus	Elementtiominaisuus
	Komponenttiominaisuus
	Direktiiviominaisuus
		`<img [src] = "heroImageUrl">`
		`<hero-detail [hero]="currentHero">`
		`</hero-detail>`
		`<div [ngClass] = "{selected:`
		`isSelected}"></div>`

Tapahtuma	Elementtitapahtuma
	Komponenttitapahtuma
	Direktiivitapahtuma
		`<button (click)="onSave()">`
		`Save</button>`
		`<hero-detail (deleteRequest)`
		`= "deleteHero()"> </hero-detail>`
		`<div (myClick)="clicked=$event">`
		`click me</div>`

Kaksisuuntainen	Tapahtuma Ominaisuus	`<input [(ngModel)]="heroName">`
Attribuutti	Attribuuttipoikkeus	`<button [attr.aria-label] = "help">help</button>`
Luokka	Luokkaominaisuus	`<div [class.special] ="isSpecial">Special</div>`
Tyyli	Tyyliominaisuus	`<button [style.color] = "isSpecial ? 'red' : 'green'">`

3.6 Riippuvuusinjektorit

Riippuvuusinjektorin käsite on lähdekoodin kirjoittamisen tapa, jossa ulkopuolisesta lähteestä voidaan rakentaa riippuvuuksia ohjelmaluokkiin ja injektoida niiden sisältämää dataa ohjelmaluokan muuttujiin. Angularissa nämä riippuvuudet ovat sisäänkirjoitettuja sen lähdekoodiin, ja Angular pitää myös huolta riippuvuussuhteiden elinkaarista. Angular luo ja kutsuu injektoreita, kun se lukee komponentteja joko HTML-merkkikielen tai reitittimen määrittämistä valitsijoista.

Palvelut ovat Angularissa luokkia, jotka voidaan metadatan avulla rekisteröidä käyttämään injektoriominaisuutta. Koodiesimerkissä 5 on esimerkkiluokka, joka toimii injektoripalveluna.

```
import {Injectable} from 'angular2/core';
import {Hero} from './hero';
import {HEROES} from './mock-heroes';
import {Logger} from '../logger.service';

@Injectable()
export class HeroService {
   constructor(private _logger: Logger) { }
   getHeroes() {
     this._logger.log('Getting heroes ...')
     return HEROES;
   }
}
```

Koodiesimerkki 5. Injektoinnin mahdollistava palveluluokka [18].

Tiedostot, jotka halutaan riippuvuussuhteeseen, on tuotu luokkaan *import*-komennoilla ja injektointiominaisuus on ilmoitettu merkinnällä *@Injectable()*. Luokassa on muodostin, joka ottaa vastaan *logger*-parametrin, joka taas on riippuvainen *logger.service* -luokasta. Kun *logger* on muodostettu, voidaan hakea *HEROES*-objekti luokasta *mock-heroes*. Tämä kuvaa esimerkkiä sisäänkirjautumisesta, jossa validoi-

daan käyttäjä ja sen jälkeen haetaan näytettävä data. Tämän jälkeen tätä dataa voidaan käyttää myös ohjelman lapsiluokissa ja esimerkiksi muokata sen sisältämiä informaatio-osasia.

Riippuvuus on siis kytketty ja injektori pitää huolen dataelementtien luovutuksesta muihin luokkiin. Injektoriominaisuus muodostuu jo Angular-sovelluksen käynnistyessä ja on oletuksellisesti aina käytössä. Koodiesimerkin 5 tuottama injektoitava tieto täytyy määritellä ohjelman muissa luokissa tarvittaessa *@Component*in sisällä, joka on nähtävillä myös koodiesimerkissä 4. Tässä tapauksessa siihen lisätään taulukko *providers:[HeroService, Logger]*. Samaan luokkaan on lisättävä myös muodostin (*constructor*), joka ottaa vastaan tarvittavan tiedon, kuten koodiesimerkissä 6.

```
constructor(heroService: HeroService) {
    this.heroes = heroService.getHeroes();
}
```

Koodiesimerkki 6. *HeroService*-muodostin.

3.7 Http-palvelinpyynnöt ja JSON-objektien käyttö

Angular 2.0:aan on sisäänrakennettu injektoitava luokka, jonka avulla voidaan suorittaa http-pyyntöjä. Kutsu luokasta palauttaa havaintosuureen (engl. observable). Angular käyttää ReactiveX:n kehittelemää *Observable*-kirjastoa, joka antaa asynkronisen mahdollisuuden käyttää peruttavia tarkkailuarvoja tapahtumajonoista. Koodiesimerkissä 7 nähdään palveluluokkakomponentti, joka tekee *get*-metodilla verkkopyynnön ohjelmointirajapinnasta, joka taas palauttaa JSON-objektin.

```
import {Injectable}     from 'angular2/core';
import {Http, Response} from 'angular2/http';
import {Hero}           from './hero';
import {Observable}     from 'rxjs/Observable';

@Injectable()
export class HeroService {
  constructor (private http: Http) {}

  private _heroesUrl = 'app/heroes'; // URL to web api

  getHeroes () {
    return this.http.get(this._heroesUrl)
           .map(res => <Hero[]> res.json().data)
           .catch(this.handleError);
```

```
  }

addHero (name: string) : Observable<Hero> {
    let body = JSON.stringify({ name });
    let headers = new Headers({ 'Content-Type':
          'application/json' });
    let options = new RequestOptions({ headers:
          headers});
    return this.http.post(this._heroesUrl, body,
          options)
          .map(res => <Hero> res.json().data)
          .catch(this.handleError);
  }

 private handleError (error: Response) {
    console.error(error);
    return Observable.throw(error.json().error ||
      'Server error');
  }
}
```

Koodiesimerkki 7. Http-palvelukomponentti [19].

Muodostin ottaa vastaan yksityisen http-objektin. Kun funktio *getHeroes()* suoritetaan, se palauttaa *http.get* -metodin kutsulla ohjelmointirajapinnan osoitteesta vastausobjektin (JSON) ja muodostaa vastauksesta havaintosuureen, joka luo *Hero*-objektin, josta saadaan näytettävä informaatio sivustolle. Funktio *addHero()* ottaa vastaan havaintosuureen ja lähettää palvelimelle http-pyynnön informaation lisäämiseen. Pyynnössä (*post*) huomioidaan http-protokollan tarvittavat ylätunnisteet ja tieto käännetään JSON-muotoiseksi. Tarvittavat virheenkäsittelijät huomioidaan tässä kirjoittamalla ne vain esimerkin testausta varten konsoliin.

4 Verkkosivun muutokset ja ohjelmistokehyksen kehittyminen

Insinöörityön aihe kehittyi tarpeesta päivittää aikaisemmin tekemäni blogi-tyyppinen verkkosivusto ja sen hallintapaneeli. Vanhalla sivustolla oli käytetty ohjelmointikielinä HTML:n ja CSS:n lisäksi PHP:tä, JavaSriptiä ja jQuerya. Sivuston sisältämä data ladattiin palvelimelta MySQL-taulukkosta ja sivustolla oli useita aukenevia osioita ja lomakkeita. Osioiden sisällä oli vaihtuvaa dataa ja riippuen siitä mitä klikattiin, data päivitettiin yleensä palvelimelta yksittäisinä hakuina. Tarkoitus oli lähteä toteuttamaan tämän sivuston päivitystä nykyaikaisempaan ja dynaamisempaan suuntaan ja niin, ettei jokai-

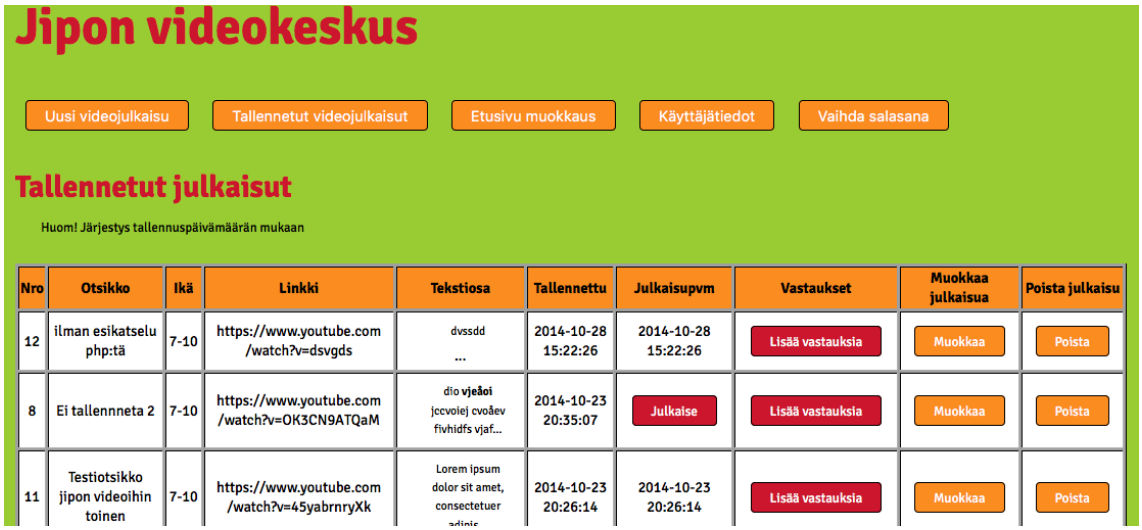
sessa käyttäjän syötteestä ja niiden tarkastamisesta jouduttaisi lataamaan sivuston sisältöä aina kokonaan uudelleen. Angular tuntui sopivalta kehykseltä tähän työhön.

Koska Angularista oli tarjolla kaksi versiota, päädyin tutustumaan uudempaan versioon lähemmin ja kokeilemaan, olisiko se käyttökelpoinen ohjelmoimalleni sivustolle. Näin sivuston päivitys tulisi jo tehtyä valmiiksi uudemmalla versiolla. Vanhemmilla 1.x -versioilla todennäköisyys on kuitenkin, että sivustoa pitäisi päivittää taas uudelleen kun Angular 2.0 -versio alkaa olla lopullisessa julkaisuvaiheessaan ja enemmän käytössä ja kun 1.x -versioita ei enää ylläpidetä. Uuden version julkaisupäivää ei kuitenkaan ole vielä Angular-tiimin taholta annettu, mutta todennäköistä on, että jo tämän vuoden (2016) puolella beetaversiosta siirrytään viralliseen käyttöversioon. Ohjelmistokehyksen uuden version dokumentaatiokin on sillä mallilla, että sitä pystyy hyvin oman työni kaltaisella yksinkertaisella sivustolla käyttämään. Useita hienojakoisempia Angular 2.0:n osa-alueita tässä päivitystyössä ei edes tarvita.

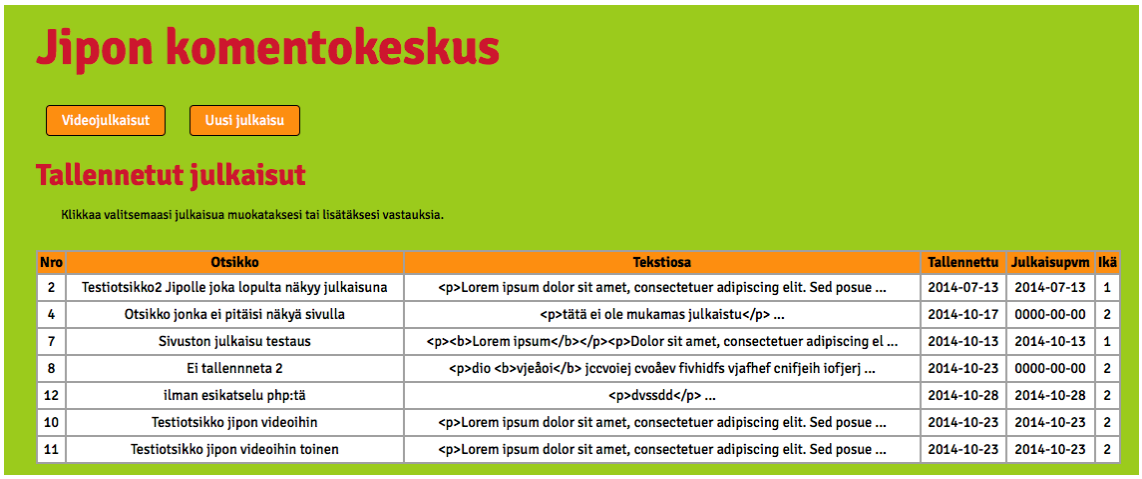
4.1 Erot vanhan ja uuden verkko-ohjelmoinnin välillä

Huomattavin muutos korjauksen alla olevalla sivustolla on erilaisten painikkeiden häviäminen käyttöliittymästä ja se, että lomakkeita täyttäessä voidaan julkaisua esikatsella reaaliaikaisesti. Tallennus tietokantaan suoritetaan vasta, kun muutokset on tehty. Käyttöliittymässä avautuvia uusia alisivustoja on vähennetty. Vaikka ohjelmallisella tasolla eri osioita onkin nyt enemmän, ne on rakennettu pienemmistä osista. Eri moduulit suoritetaan vain tarvittaessa, ja näin asiakkaan päässä käyttöliittymä on rakenteeltaan selkeämpi.

Sivulla 19 kuvassa 7 on näkymä vanhasta käyttöliittymästä, jossa on listausta blogijulkaisuista. Kuvassa on näkyvillä monia eri painikkeita, jotka käynnistävät eri tapahtumia tai avaavat uusia alisivuja. Kuvassa 8 on uusi versio, jossa nappeja nähdään tuskin lainkaan. Tässä näkymässä on jo itsessään käynnissä useita Angular 2.0:n mukaisia moduuleita. Suurin osa vanhoista napeista on tässä tarpeettomia. Uudessa liittymässä onkin vain maininta ”Klikkaa valitsemaasi julkaisua muokataksesi julkaisua tai lisätäksesi vastauksia”.

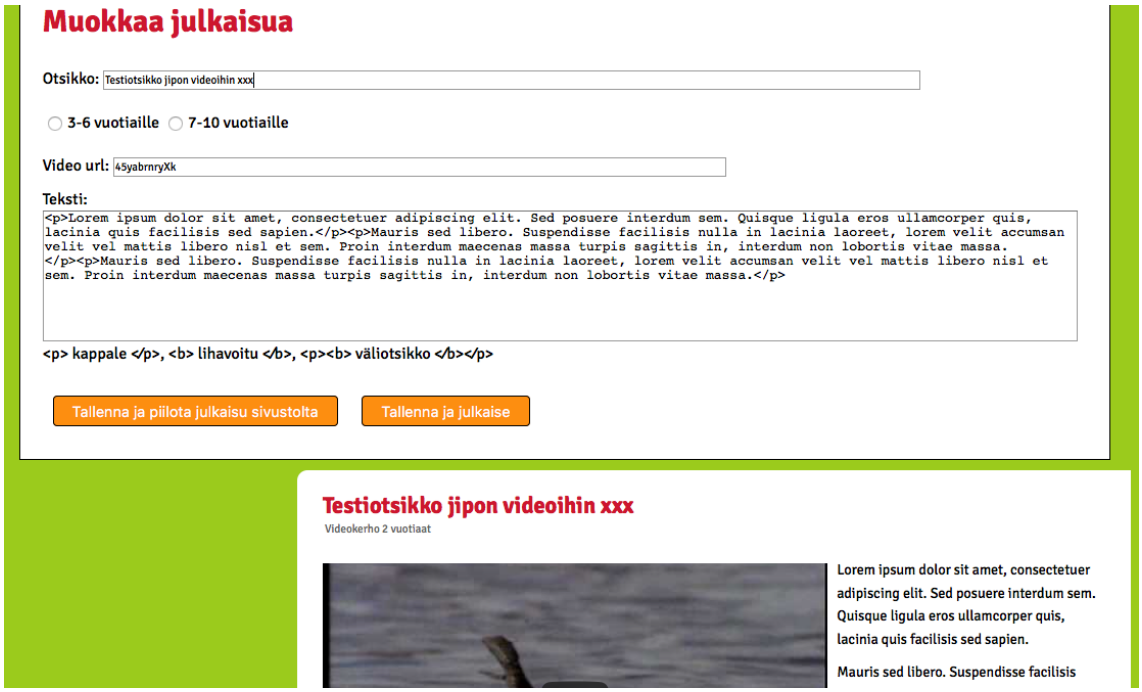


Kuva 7. Taulukkolistaus vanhassa käyttöliittymässä.



Kuva 8. Taulukkolistaus uudessa käyttöliittymässä.

Uuden käyttöliittymän myötä listauksen osiota klikatessa avautuu yksittäisen julkaisun näkymä, joka nähdään sivulla 20 kuvassa 9. Julkaisua pääsee heti muokkaamaan, ja esikatselu toimii samassa näkymässä. Esikatselu muuttuu samanaikaisesti sen mukaan, mitä lomakkeisiin kirjoitetaan. Muutosten jälkeen julkaisuosa voidaan tallentaa tietokantaan. Vanhassa sivustossa yksittäiset julkaisut avautuivat muokkausvaiheeseen ja esikatselunäkymä oli erillinen alisivu, josta edestakaisin selailemalla täytyi aina tarkastaa kokonaistulos ennen tallennusta, kuten näkyy kuvassa 10.



Kuva 9. Muokkaus ja esikatselu samalla sivulla uudessa käyttöliittymässä.

Kuva 10. Muokkaus ja esikatselu erillisinä sivuina vanhassa käyttöliittymässä.

Vanhalla sivustolla reititys tapahtui PHP:n *switch*-lausekkeilla. Jokainen alisivu oli oma osionsa. Nykyinen reititys tapahtuu Angular 2.0 -reititinmoduulien mukaisesti ja reititinosia ja ehtolausekenäkymiä suoritetaan yhtäaikaisesti. Näkymästä piilotettuja Script- ja DOM-elementtejä, jotka ovat ominaisia jQuerylle, ei käyttäjän päässä selaimeen renderöity ollenkaan. Näkymiä on uudessa versiossa rakennettu julkaisujen yksilötasolla Angularin mukaisesti. Edellisessä versiossa yksittäisten julkaisujen valitsemiseen oli käytetty PHP:llä lisättyjä sessioita.

4.2 Tietokannat ja PHP

Lomakkeiden syöttöjen tarkastus on uudella sivustolla toteutettu suurimmalta osalta Angular 2.0:n piippurakenteilla, ja niitä suoritetaan tarvittaessa erillisistä moduuleista. Uudella sivustolla jokaisesta yksittäisestä muutoksesta ei tehdä tietokantamuutoksia. Luvussa 4.4.3 kerrotaan, kuinka tiedon tallentaminen ja haku tulisi toteuttaa niin, että tämäkin klikattava tallennusvaihe jäisi pois ja tieto kulkisi jatkuvasti muutosten mukaisesti palvelimen ja käyttäjän välillä. Muutostöitä blogisivustolla ja sen hallintapaneelissa tehdään pitkällä aikavälillä, ja asteittainen siirtyminen uudenlaisiin ohjelmointimalleihin myös palvelimen puolella on otettu huomioon.

Angularille ominaista olisi käyttää tietojen hakuun jatkuvasti auki olevia WebSocket-tyyppisiä tietoyhteyksiä. Toinen vaihtoehto on käyttää RESTful APIa, joka palauttaa tiedot JSON-objektina, kuten luvun 3.7 esimerkissä on tehty. Rajapinta on rakennettu uudistetulle sivustolle niin, että Angular hakee tiedon PHP-tiedostosta, joka on palauttanut erillisen kyselyn mukaisen MySQL:n tietokantaelementit JSON-objektiksi käännettyinä. Vastaavanlaisesti suoritetaan tietokantaan lisäykset tai tietojen poistaminen aina PHP:n kautta käännettynä.

Vaikka palvelinpuolen ohjelmointi ja taulukot olisi järkevämpää rakentaa muulla tavalla, tässä päädyttiin käyttämään toistaiseksi jo olemassa olevia MySQL-taulukoita ja rakentamaan tietokantakyselyt niiden ympärille. Luvussa 4.4.3 pohditaan palvelinpuolen ohjelmoinnin nykyaikaistamista. Angular ei itsessään ota kantaa siihen, miten tämä tehdään.

4.3 Parannukset

Kuten Angularin kehittelijät ovat kertoneet [2], tämän ohjelmistokehyksen käytön myötä verkko-ohjelmoinnista on tullut monimuotoisempaa ja älykkäämpää. Erilaisia valmiita ohjelmallisia moduuleja käyttäen on saatu vähällä vaivalla dynaamisia kokonaisuuksia aikaiseksi. Moni näistä valmiista moduuleista on ohjelmoinniltaan niin korkealuokkaisia, että vastaavien rakentaminen tyhjästä olisi työlästä.

Vertailtuani Angularin 1.x ja 2.0 -versioita voisin sanoa, että tutustuttuani tähän uuteen versioon minulla ei ole enää edes kiinnostusta syventyä vanhempien versioiden yksi-

tyiskohtiin. Ne vaikuttavat turhan monimutkaisilta. Uusi versio on mielestäni alusta lähtien ollut selkeäjakoisempi ja ymmärrettävämpi. Tuntuu siltä, että vastustus tätä uutta versiota kohtaan on ollut enemmänkin huolta asioiden muuttumisesta ja ohjelmointityylien erilaisuudesta. Uudistusten tuomat muutokset ovat kiistatta parantaneet Angularin toimintoja ja laskentaa.

Angularin käytön myötä rakentamani verkkosivun rakenteellinen selkeys parani. MVC-tyyppinen erottelu näkyy tiedostorakenteessa, ja datan siirtyminen osiosta toiselle helpottui injektoreilla ilman sessiokäsittelijöitä. Voisi sanoa, että verkkokehittely elää yleisesti murroskautta. JavaScript-pohjainen ohjelmointi on kasvattanut asemaansa, ja oman sivustoni edellisessä versiossa käytetty PHP-ohjelmointi tuntuu jo vanhentuneelta, vaikka se on edelleen hallitsevassa asemassa verkkokehittelyssä.

Itse hallintapaneelin sivustolla reititys toimii selkeämmin ja lomakkeiden syötteet pystytään suurimmalta osin tarkastamaan jo niitä tehdessä. Käyttäjän näkökulmasta ehdottomasti paras muutos on ollut esikatselun näyttö yhtäaikaisesti lomakkeita täyttäessä. Monia turhia klikkauksia on karsittu, ja käyttäjän liikkuminen sivustolla on selkeytynyt.

4.4 Puutteet

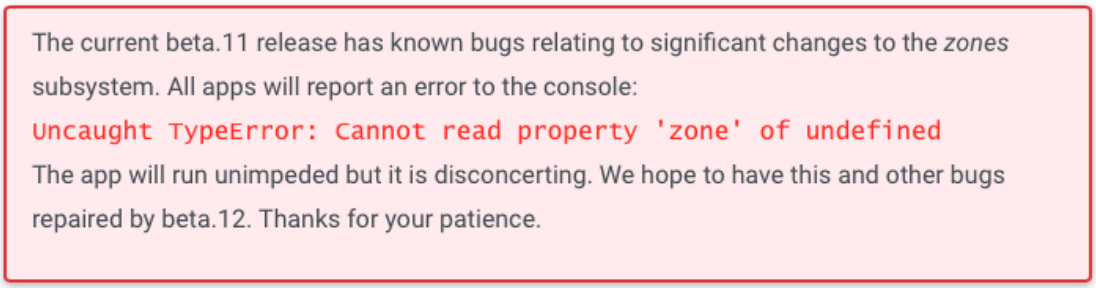
Angular 2.0 -ohjelmistokehyksen käyttöönotto ei ole kuitenkaan sujunut aivan ongelmitta. Ohjelmointimalleihin tutustuttaessa uudistuksia on tehty, tiedon välittämisessä on rakennettu kompromissiratkaisuja ja testauksessa eri selaimilla on todettu vajavaisuuksia. Näitä asioita käydään tarkemmin läpi seuraavissa alaluvuissa.

4.4.1 Päivitykset

Huomattavaa verkkosivuston työstövaiheessa oli, että tänä aikana Angularin node-moduulikirjastot päivitettiin useaan otteeseen. Aikaisemmat harjoitteet eivät enää selaimessa toimineet sellaisenaan, vaan uudet kirjastot täytyi ladata ja asentaa. Yhdessä päivitysvaiheessa myös Angular 2.0:n tutoriaalisivusto oli täysin uudistettu. Koodikielen toiminta tosin ei ollut kuitenkaan huomattavasti muuttunut, ja uusia ohjeita ja malliesimerkkejä löytyi tuon uudistuksen myötä enemmän. Myös lopullisen julkaisun myötä voidaan olettaa, että päivityksiä tulee useasti jatkossakin ja valmiudet kirjastojen uusi-

miseen omalla sivustolla täytyy pitää yllä, ettei sivusto olisi kauan haavoittuvainen tai toimintakyvyttömänä.

Kuvassa 11 nähdään Angular 2.0:n sivustolla näkyvä ilmoitus ohjelmointivirheistä vielä maaliskuun 2016 loppupuolelta ja samalla GitHub-versionhallintajärjestelmässä ilmoitettiin uusi avoin virstanpylväs beeta 12 -version julkaisusta [20]. *Zones* on tärkeä osa Angularin koodistossa: se tarkastaa arvojen muutoksia ohjelmistossa. Kuitenkin jo seuraavana päivänä kirjasto oli korjattu ja varoitukset olivat hävinneet. Nodemoduulikirjastot taas olivat uudestaan ladattavissa. Virheitä korjataan siis nopeasti.



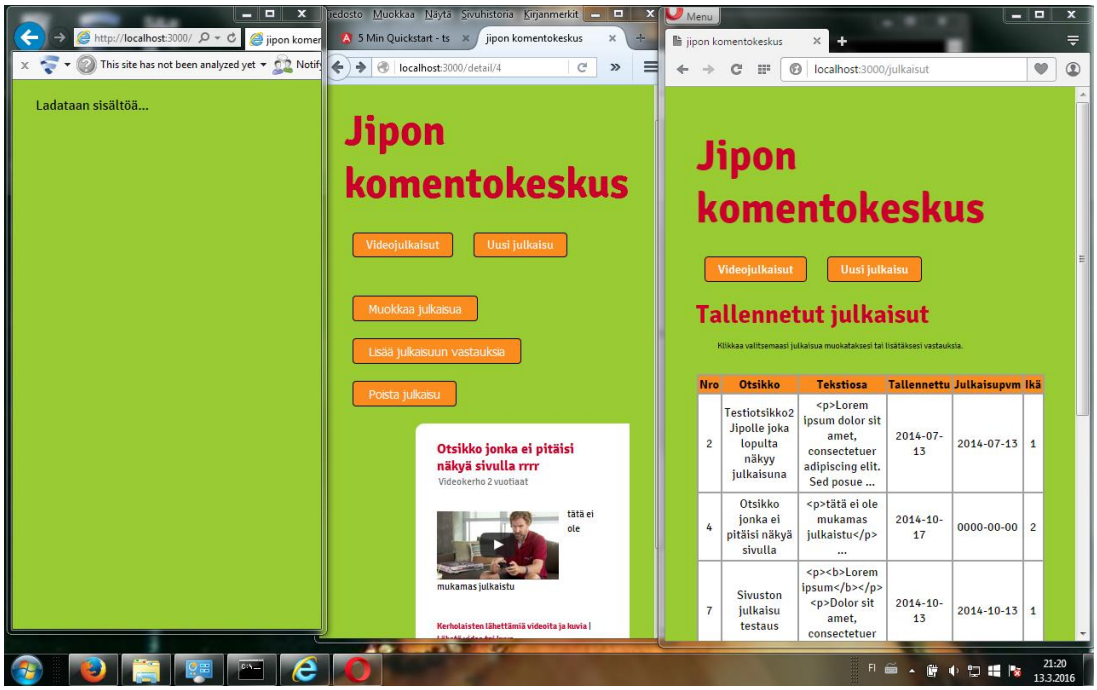
Kuva 11. Ilmoitus kirjastojen ohjelmointivirheistä [21].

4.4.2 Toimivuus eri selaimilla

Angular 2.0 ei ole vielä vakaa, ja kaikilla selaimilla, varsinkaan vanhemmilla versioilla, se ei toimi. Päivityksen alla olevan sivuston käyttäjäkunnassa on paljon Internet Explorer -selainten (IE) käyttäjiä, sillä Helsingin yliopiston, jolle tätä sivustoa on rakennettu, tietokoneilla oletusselain monesti on juuri tämä Microsoftin julkaisema vaihtoehto. Vain harjaantuneemmat koneenkäyttäjät ovat ymmärtäneet vaihtaa selaimen muuhun, paremmin toimivaan vaihtoehtoon. Kotikäytössä taas voi olla hyvinkin erilaisia selaimia ja laitteita, joilla sovellusta mahdollisesti käytetään. Tunnetusti IE-selainmalli ei ole ollut toimiva muidenkaan uusien verkkostandardien kanssa. Angular 2.0:n toimivuus taataan täysin ainoastaan Googlen Chrome-selaimella, silti se toimii myös muilla suurimmilla selaimilla. Sivulla 24 taulukossa 3 nähdään, että omassa testissäni ainoastaan Internet Explorer -selain ei toiminut. Tästä havainnollistuksena myös kuva 12.

Taulukko 3. Testitulokset Angular 2.0:n toimivuudesta eri selaimilla ja käyttöjärjestelmillä.

Mac	Chrome	✓
	Mozilla Firefox	✓
	Safari	✓
Windows	Internet Explorer	✗
	Mozilla Firefox	✓
	Opera	✓



Kuva 12. Sovelluksen toimintatestaus eri selaimilla: Internet Explorer-, Mozilla Firefox- ja Opera-selainnäkymät.

4.4.3 Tiedon haku palvelimelta

Parannettavaa itse sivuston päivityksessä on tietokantojen hakemisessa. PHP ja MySQL eivät anna oikeuksia Angular-tyyppiseen ohjelmoimiseen. Tämä PHP-ohjelmointi toimii, mutta on välillä turhan monisyistä. Samoja asioita toistetaan niin palvelimen kuin asiakaspuolen päässä, ja tiedot käännetään puolelta toiselle MySQL-kannasta JSON-objeteiksi ja toisinpäin. Mikäli aikaa jää, ennen sivuston lopullista päivi-

tysjulkaisua uuden palvelinpuolen kunnollinen ohjelmointirajapinta, joka ei käyttäisi lainkaan SQL-taulukoita, voisi olla vielä toteutettavissa, tai vastaavasti WebSocket-tyyppinen ohjelmointi olisi omiaan tiedonvälittämiseen tällä sivustolla.

Monessa yhteydessä on suositeltu Angularin kanssa käytettäväksi Node.js-ohjelmointikieltä tiedonsiirtoon palvelimelta. Node.js pystyy kuuntelemaan http-palvelupyyntöjä rinnakkaisesti, ja tämä vauhdittaa tiedonhakua päivittämättä aina koko sivustoa, toisin kuin PHP. Informaatio palvelimella muuttuu käyttäjän syötteiden mukaisesti lähes reaaliajassa. Palvelinpuolen tiedonvälittämiseen on myös olemassa muita ohjelmistokehyksiä. Etenkin Firebase-niminen JSON-objekteja välittävä kehys on tässä tutkimustyössä tullut esiin. Firebase on kuitenkin maksullinen ohjelma. Sitä on suositeltu myös tiedonsiirtoon mobiiliohjelmille ja niiden natiiviohjelmointiin, johon myös Angular 2.0 voidaan sulauttaa. Mobiiliohjelmoinnista kerrotaan lisää luvussa 4.5.2.

4.5 Analyysi ja pohdintoja

Oma kokemus Agular 2.0 -ohjelmistokehyksen käyttöönotosta on ollut hyvin positiivinen, ja ohjelmointityyli on mielestäni selkeää. Kuten sanottu, ohjelmallinen nopeus ei ole itselleni tässä vaiheessa ollut tärkeää. Rakentamani sivustot eivät ole olleet niin raskaita tai käyttäjäkunta laaja, että sillä olisi merkitystä. Moduulien välisten osasten sulavat kytkökset ja muovautuminen eri käyttöön ovat kuitenkin olleet tervetulleita ominaisuuksia verkkokehitystyöskentelyyn. Päivityksen myötä kirjoitettu kokonaiskoodin määrä kasvoi, mutta toiminnalliset komponentit ovat erillisiä ja helposti muokattavissa. Vain tarvittavat osat ovat käytössä yhtä aikaa asiakaspäässä, eikä koko koodisto kerralla.

Kuten luvussa 4.4.1 kerrottiin, myös tämän insinöörityön aikana Angular 2.0 -kirjastoa päivitettiin, ja yleisenä neuvona voisi pitää sitä, että kannattaa odottaa uuden version käyttöönotossa. Päivityksiä voi jo nyt alkaa rakentaa asteittain niin sanottuna testialusversiona. Tämänkään insinöörityön pohjalla olevan sivuston päivityksillä ei ole määritelty varsinaista julkaisuajankohtaa. Vanhat sivut ovat kovassa käytössä, ja uusia versioita testataan asteittain.

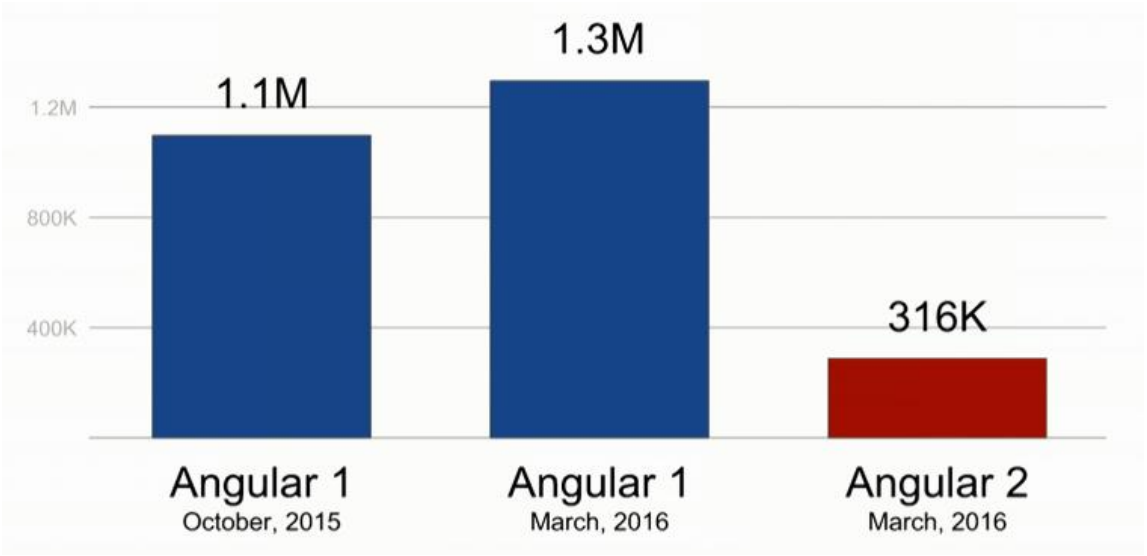
Angular 2.0:lla rakennettu sivusto on tarkoitus julkaista blogisivuston hallintapaneelin osalta, ennen kuin ulkopuolelle näkyvää sivustoa vielä uudistetaan. Tällöin voidaan

käytännössä testata kehyksen toimivuutta, ja siitä ei ole haittaa oikean sivuston asiakaskäyttäjille. Myöhemmässä vaiheessa myös itse blogisivustoa olisi tarkoitus uudistaa. Mielestäni Angular 2.0 on käyttökelpoinen tuohonkin päivitykseen, kunhan uusi versio stabiloituisi. Luottamusta ohjelmistokehyksen lopulliseen toimivuuteen itselläni kyllä on.

4.5.1 Tulevaisuus

Toukokuun 2016 alussa pidetään Angularin tiimoilta konferenssi Salt Lake Cityssä, Yhdysvalloissa. Myös syys- ja lokakuussa on kokoontumisia pelkästään Angular-aihepiirin tiimoilta Euroopassa. Vaikka beetaversiosta siirtymisestä ei ole määritelty muutoin kuin ilmoittamalla vielä saavuttamattomia virstanpylväitä, näistä kokoontumisista voidaan odottaa uutta tietoa lopullisesta julkaisusta. Angularin GitHub-versionhallintajärjestelmän sivuilla on ilmoitettu, että julkaisua edellyttävät viimeisimmät ongelmakohdat olivat 14.3.2016 kahdeksanprosenttisesti ratkaistu ja jo muutaman päivän kuluttua 17.3.2016 prosentti oli 21 [20].

Angular 2.0:n alfavaiheen julkaisun jälkeinen epäröinti verkkokeskusteluissa on hiipunut ja uusia oivalluksia ja käyttökokemuksia jaetaan paljon. Kuvassa 13 nähdään, että Angularin 2.0 -versiota käytetään jo ahkerasti samalla kun 1.x -versioiden käyttö on myös jatkanut kasvuaan.



Kuva 13. Agularin versioiden käyttö maaliskuussa 2016 [22].

Ratkaisuja ohjelmointikysymyksiin on silti vielä vaikeaa löytää yhteisöllisiltä foorumeilta. Esimerkiksi Stack Overflow -verkkosivustolla ohjelmointikysymyksiä oli erilaisilla hakutuloksilla seuraavanlaisesti:

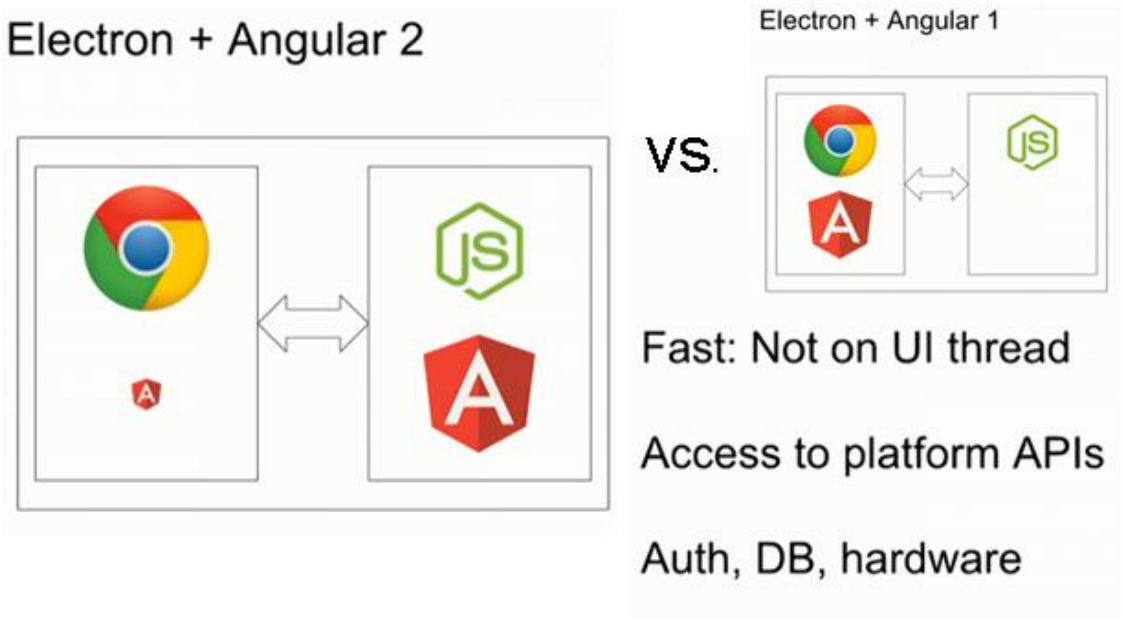
- angular 2: 13 895 tulosta
- angularjs: 159 820 tulosta
- angular: 145 983 tulosta
- react: 44 470 tulosta
- ember: 24 219 tulosta [23].

Kuitenkin myös YouTube-videosivustolla on joka päivä uusia koodiesimerkkejä ja videotutoriaaleja Angular 2.0:n käyttöönotosta. Kasvu on näkyvillä myös jo sivulla 7 nähdyssä viivakaaviossa Googlen hakutuloksista (kuva 3). Vaikka 2.0 -version verkkohakumäärä onkin vielä pientä, se on lähtenyt kasvuun, ja voidaan olettaa, että pelkkää angular-hakutermiä käytetään myös molempien versioiden hakuun. Angular 2.0 -versiosta on myös tekeillä ohjelmointikirjoja.

4.5.2 Mobiilisovellukset

Angular on nykyaikainen ohjelmistokehys, jolla voidaan rakentaa dynaamisia ja nopeita käyttöliittymiä. Tämä on edellytys nykypäivän aina vain kasvavassa mobiilissa verkkokäytössä [24]. Angular 2.0 tekee mahdolliseksi käyttää iOS:n ja Androidin natiivia mobiiliohjelmointia sen sisällä. Angular-tiimi tekee paljon yhteistyötä NativeScript- ja Ionic-ohjelmistokehyksien kanssa, ja se mainostaa sivustoillaan, että näitä käyttämällä yhdessä syntyy kevyt ja nopea hybridikäyttöliittymä mobiilisovelluksiin riippumatta raskaasta datamäärästä taustalla [25].

Yhteistyöhankkeista voi kehittyä käyttökelpoisia ohjelmistokehyskokonaisuuksia, jotka tarjoavat niin asiakaspään kuin palvelinpuolen verkko-ohjelmointiin työkalut varmasti ja helposti. Sivulla 28 kuva 14 havainnollistaa, miten Angular 2.0 toimii suhteessa selainpäässä ja palvelimen puolella ja kuinka tietoa voidaan välittää käyttäjälle verrattuna aikaisempaan versioon. Käyttäjäpäässä ohjelmallinen taso on pientä ja kevyttä, ja suurin osa ohjelmasta suoritetaan laskennallisesti jo palvelimen puolella esilatausohjelmallisuudella [26]. Esimerkki on Electronin ja Angularin yhteistyöstä.



Kuva 14. Asiakas-palvelinarkkitehtuurimalli kehitteillä olevasta Angularin ja Electronin yhteistyökäytöstä [22].

Tämänkaltaisen ohjelmoinnin pitäisi olla nykypäivää ja tulevaisuuden verkkoohjelmoinnin perusta. Vaikka tekniikka kehittyy ja laitteiden laskenta kasvaa, pienillä ja mobiileilla laitteilla verkkoselaaminen on erilainen kokemus, ja se tulisi ottaa huomioon kaikkia verkossa olevia ohjelmistoja ja sivustoja kehiteltäessä.

5 Yhteenveto

Angular 2.0 on tulevaisuuden teknologioihin ja standardeihin perustuva ohjelmistokehys, joka on luotu helpottamaan älykästä ja dynaamista verkkokehittelyä. Angularin vanhemmat versiot ovat jo vanhentuneita, ja uuteen versioon on tehty perustavanlaisia ohjelmallisia muutoksia. Angular on suosituin ohjelmistokehyksistä kilpailijoihinsa nähden, ja uudesta versiosta ennustetaan sen seuraajaa. Angular 2.0 on vasta beetavaiheessa ja varsinaista julkaisupäivämäärää odotellaan vielä vuoden 2016 puolella.

Insinöörityössä tutkittavaksi valittiin ohjelmistokehyksen uusi 2.0 -versio, jonka avulla jo olemassa oleva verkkosivusto ja sen hallintapaneeli saataisiin päivitettyä nykyaikaisempaan muotoon. Näin sitä ei olisi myöskään tarvetta muuttaa uudestaan Angularin 1.x -versioiden päivitysten lakatessa. Vanha sivusto, joka on pääosin PHP:llä rakennettu, on kovassa käytössä. Uudistusta on aikaa kehitellä ja testailla ja koodiesimerkkejä

ja ohjeistusta Angular 2.0:sta oli jo saatavilla tarpeeksi, jotta tätä sivuston päivitystyötä voitiin alkaa rakentaa. Matkan varrella ohjeistuksia julkaistiin vielä lisää, ja ohjelmointivirheitä moduulikirjastoista on ratkaistu ja päivitetty useaan otteeseen.

Insinöörityössä perehdyttiin Angular 2.0:n ohjelmalliseen arkkitehtuuriin muutostöiden vaatimilla alueilla ja samalla sen perusohjelmallisuuksiin tehtiin yleiskatsaus. Moduulien toimintoja testattiin käytännössä ja todettiin, että sivuston hallintapaneelin muutoksissa tehtiin parannuksia sivuston yleiseen käyttöliittymään. Monet turhat nappulat ja alisivujen lataamiset palvelimelta karsittiin pois. Lomakkeiden täyttö on luotettavampaa, ja julkaisujen muokkaamisessa esikatselu toimii reaaliaikaisesti.

Vaikka Angular ei ota kantaa palvelinpuolen ohjelmointiin, todettiin, että käytännöllinen ratkaisu olisi rakentaa se jatkuvasti auki olevilla tietoyhteyksillä, esimerkiksi Node:js-ohjelmointikieltä käyttäen. Näin tietoa voitaisiin päivittää palvelimelle jatkuvasti. Tämä vaatisi lisätyötä verkkosivuston päivitysten kannalta, sillä tietokannat on jo valmiiksi rakennettu MySQL-taulukoin, joten nyt päätettiin rakentaa PHP:llä erillinen ohjelmointirajapinta, joka palauttaa Angularille tyypillisempää tiedostomuotoa olevan JSON-objektin. Tietokantamuutokset tapahtuvat erillisillä edelleenkyselyillä PHP:n avulla. Todettiin kuitenkin, että asteittain tapahtuvan sivuston päivityksen myötä voidaan palvelinpuolen ohjelmointi ratkaista myöhemmin.

Jo tehtyjä muutoksia testailtiin eri alustoilla ja selaimilla. Ongelmia tuotti Microsoft Explorer -selain, ja pohdintana huomioitiin, että Angular 2.0:n käyttöönottoa kannattaa hieman jarrutella, kunnes beetaversiosta on päästy ja sen jälkeiset ohjelmointivirheet ja parannukset on ohjelmointikehykseen tehty. Rakentamista voi ja kannattaakin kuitenkin jo aloittaa nykyisellä beetaversiolla.

Angular 2.0 ohjelmistokehyksellä on paljon tulevaisuutta, ja se tuo monia ratkaisuja esimerkiksi mobiilikehitykseen. Angularia ei tarvitse käyttää itsenäisesti, vaan useasta kehyksestä ja tietoteknisestä kirjastojen yhdistelmästä voidaan rakentaa kevyitä ja nopeita tulevaisuuden verkkosivustoja. Sivustoja ei kannata rakentaa pelkästään tätä päivää ajatellen, vaan myös tulevaisuutta silmällä pitäen.

Lähteet

1 Hevery, Miško. 2010. Angular: A Radically Different Way of Building AJAX Apps. Verkkovideoluento. GoogleTechTalks, YouTube Inc. <https://www.youtube.com/watch?v=elvcgVSynRg>. Päivitetty 29.7.2010. Katsottu 7.2.2016.

2 Hevery, Miško, Green, Brad. 2014. Miško Hevery and Brad Green - Keynote - NG-Conf 2014. Verkkovideoluento. ng-conf, YouTube Inc. <https://www.youtube.com/watch?v=r1A1VR0ibIQ>. Päivitetty 16.1.2014. Katsottu 8.2.2016.

3 Angular.io. 2016. Verkkodokumentti. GitHub, Inc. <https://github.com/angular/angular.io/network/members>. Luettu 8.2.2016.

4 Angular.io. 2016. Moving the Web Forward - The Angular Core Team. Verkkodokumentti. Google, Inc. <https://angular.io/about/>. Luettu 8.2.2016.

5 Green, Brad, Minar, Igor. 2015. Verkkodokumentti. Ng-conf 2015 - Day 1 Keynote. <https://docs.google.com/presentation/d/1d03YJ1gKhMZkV-87m9lsS_gnXASVDdSfYvuvzHIro6g/pub?start=false&loop=false&delayms=3000&slide=id.g7b5b0dc77_318>. Luettu 22.2.2016.

6 Minar, Igor. 2015. An Angular2 Todo App: First look at App Development in Angular2. Verkkovideoluento. AngularJS, YouTube Inc. <https://www.youtube.com/watch?v=uD6Okha_Yj0>. Päivitetty 11.2.2015. Katsottu 10.2.2016.

7 Index Of. 2016. Verkkodokumentti. AngularJS. <https://code.angularjs.org>. Luettu 14.3.2016.

8 Avoimet työpaikat. 2016. Verkkohaku. Työ- ja elinkeinoministeriö. <http://www.mol.fi/tyopaikat/tyopaikkatiedotus/haku/hae.htm?lang=fi&hakusana=angular>. Haku tehty 10.2.2016.

9 McGinnis, Tyler. 2016. Front End Newsletter. Verkkodokumentti. Front End Newsletter. <http://frontendnewsletter.com/issues/1>. Luettu 18.3.2016.

10 Google Trends. 2016. Verkkohaku. Google inc. <https://www.google.fi/trends/explore#q=angular%2C%20angular%202%2C%20angularjs%2C%20ember%2C%20react&cmpt=q&tz=Etc%2FGMT-3>. Haku tehty 10.3.2016.

11 Architecture Overview. 2016. Verkkodokumentti. Angular. <https://angular.io/docs/ts/latest/guide/architecture.html>. Luettu 17.2.2016.

12 Displaying Data. 2016. Verkkodokumentti. Angular. <https://angular.io/docs/ts/latest/guide/displaying-data.html>. Luettu 21.2.2016.

13 Forms. 2016. Verkkodokumentti. Angular. <https://angular.io/docs/ts/latest/guide/forms.html>. Luettu 29.2.2016.

14 Routing & Navigation. 2016. Verkkodokumentti. Angular. <https://angular.io/docs/ts/latest/guide/router.html>. Luettu 2.3.2016.

15 Minar, Igor. 2012. Igor Minar (Public). Verkkodokumentti. Google+. <https://plus.google.com/+IgorMinar/posts/DRUAkZmXjNV>. Julkaistu 19.7.2012. Luettu 22.2.2016.

16 Eisenberg, Rob. 2015. ANGULARJS 2.0 features and beyond. Verkkovideoluento. TechTalk, YouTube Inc. <https://www.youtube.com/watch?v=odY7fUjI1ZU>. Päivitetty 19.6.2015. Katsottu 22.2.2016.

17 Binding Syntax: an Overview. 2016. Verkkodokumentti. Angular. <https://angular.io/docs/ts/latest/guide/template-syntax.html#!#binding-syntax>. Luettu 1.3.2016.

18 Dependency Injection. 2016. Verkkodokumentti. Angular. <https://angular.io/docs/ts/latest/guide/dependency-injection.html>. Luettu 22.2.2016.

19 Http Client. 2016. Verkkodokumentti. Angular. https://angular.io/docs/ts/latest/guide/server-communication.html>. Luettu 23.3.2016.

20 Angular 2 Release Candidate. 2016. Verkkodokumentti. Angular, GitHub Inc. <https://github.com/angular/angular/milestones>. Luettu 14.3.2016, 17.3.2016 ja 23.3.2016.

21 Angular Docs. 2016. Verkkodokumentti. Angular. <https://angular.io/docs/ts/latest/>. Luettu 23.3.2016.

22 Green, Brad. 2016. Angular 2 and the future of HTML5 apps. Verkkovideoluento. O'Reilly. <https://www.oreilly.com/ideas/angular-2-and-the-future-of-html5-apps>. Katsottu 14.3.2016.

23 Search. 2016. Verkkohaku. Stack Overflow. <http://stackoverflow.com/search?q=angular+2>. Luettu 23.3.2016.

24 Internetin käyttö mobiilia, laitteet henkilökohtaisia. 2015. Verkkodikumentti. Tilastokeskus. <http://tilastokeskus.fi/til/sutivi/2015/sutivi_2015_2015-11-26_tie_001_fi.html>. Luettu 19.3.2016.

25 Features & Benefits. 2016. Verkkodokumentti. Angular. <https://angular.io/features.html>. Luettu 19.3.2016.

26 Whelpley, Jeff, Stapleton, Patrick. 2015. Full Stack Angular 2 – Jeff Whelpley and Patrick Stapleton. Verkkovideoluento. AngularConnect, YouTube Inc. <https://www.youtube.com/watch?v=MtoHFDfi8FM>. Päivitetty 20.10.2015. Katsottu 1.4.2016.